An Analysis of Github Activity for Bad Smell Identification

Rohit Nambisan
North Carolina State University
Email: rnambis@ncsu.edu

Bhushan Thakur
North Carolina State University
Email: bvthakur@ncsu.edu

Zachery Thomas
North Carolina State University
Email: zithomas@ncsu.edu

***Abstract*—Software engineering as a task is fairly free form in terms of determining the overall completeness of a given assignment. It appears to be even more difficult to determine when a project has hit a roadblock or is in trouble. In order to better explore Bad Smell detection and the factors that may lead to bad smells, we are going to analyze three projects from the NCSU Spring 2017 Software Engineering class.**

I. Introduction

With this project, we chose to analyze the Github contents of the Spring 2017 semester's Software Engineering class. We ended up analyzing our own repository as well as the repositories of two of the groups with the most logged Issues. Knowing the format and due dates of the overall task as well as having first hand experience with the overall status for each of the projects gives valuable information that we used to analyze the overall cohesiveness of each group.

The goal of this analysis is to identify features that can be extracted as tags and then group these tags into categories of Bad Smells. Projects that have a large number of bad smells are more prone to problems in development.

II. Information Collection

In order to collect the issue and commit information from the Github repositories, the Github REST API suite was utilized. There were three main calls that we used to retrieve information from the various repositories. The information returned from these calls were stored as CSV files for easy offline access and manipulation using Python and Spreadhseet tools.

A. Issue API

Issues were gathered using the issue api. Issues by default come with a lot of useful information including the issue creation time, associated milestones (assuming those milestones still exist within the repository), comments, the user who created the issue, who the issue is assigned to and when the issue was closed.

```
https://api.github.com/repos/*/issues?

[where * is the repo name]
```

B. Commit API

In order to analyze the trends in commits over the scope of the project, we had to use the Commit API. Commits are a little lighter on the details when compared to how much information issues hold. Commits store useful information about who made the commit, the commit message, and the time that the commit was made. Unfortunately there is no information bout the actual content pushed with a commit. There is no information about the number of lines of code added or deleted or the names of files that were modified, added or deleted.

```
https://api.github.com/repos/*/commits?

[where * is the repo name]
```

C. Milestone API

Even though milestones appear in the Issue API, all milestones are not grunted to be present. Any milestone that does not have associated issues will not show within the Issue API call. Therefore, in order to retrieve information about all milestones regardless of status, the Milestone API was used.

The Milestone API was also used to gather information that complements the data collected by the Commit API.

```
https://api.github.com/repos/*/milestones?

[where * is the repo name]
```

III. Feature Detection

Using the information gathered from the various Github REST APIs we used, processing was implemented in order to view the data in an easy to digest format. Using trends in data and prior knowledge from our own development cycle, we can from conclusions about which features signify bad smells.

A. Issues

1) Time between opening and closing of each issue: This metric is use in order to determine the overall number of days passed between the opening and closing times of an issue. Issues are flexible in use and thus and the distribution of timing information reflects this. Group 2 tends to close issues extremely early while Groups 0 and 1 have a more scattered distribution of number of days required to close an issue.

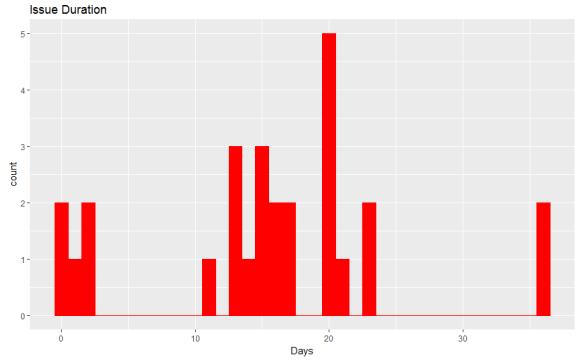

Fig. 1. Issue close time distribution for Group 0

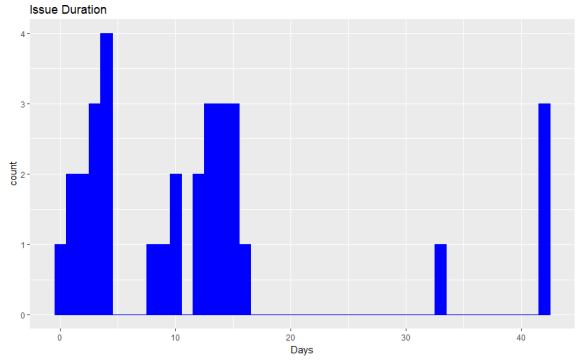

Fig. 2. Issue close time distribution for Group 1

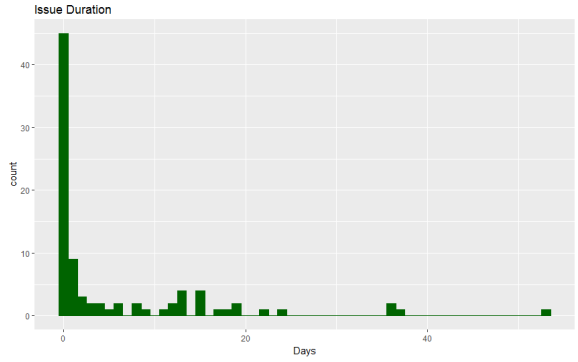

Fig. 3. Issue close time distribution for Group 2

2) Time between due date and closing of milestone: This metric measures the overall number of days that milestones are closed early or late. Milestones that have a due date but are never closed do not count within this metric and are disregarded.

Group 1 has an interesting plot as they only had one milestone which they closed. Group 2 has what seems like a fairly healthy pattern of closing some milestones fairly early while finishing other milestones a few days after the due date. Group 0 seemed to have set milestones early on, but disregarded them until the end of the development cycle.

3) Issue created and closed by different users: It is interesting to see which particular users ended up managing issues

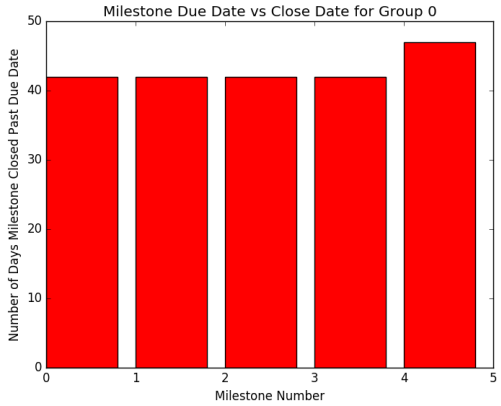

Fig. 4. Plot of Group 0 detailing the number of days between the due date of a milestone and the closing of the milestone

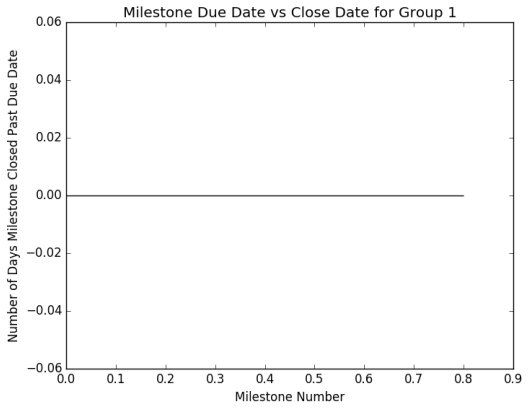

Fig. 5. Plot of Group 1 detailing the number of days between the due date of a milestone and the closing of the milestone

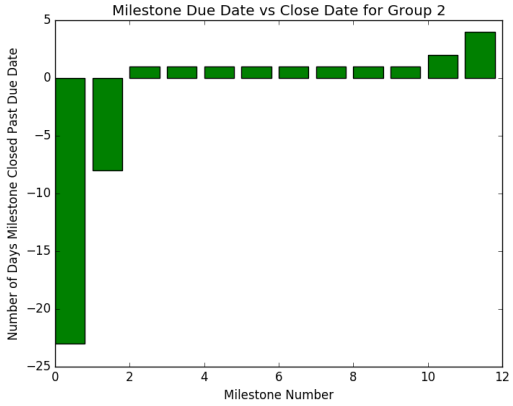

Fig. 6. Plot of Group 2 detailing the number of days between the due date of a milestone and the closing of the milestone

who were not the original issuer. Overall a decent percentage of issues closed by a user other than the issuer may indicate a healthy team cohesion.

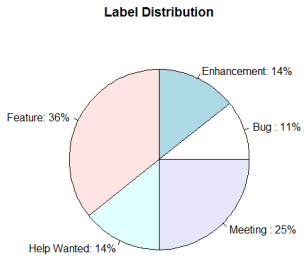

Fig. 7. Label Distribution for Group 0

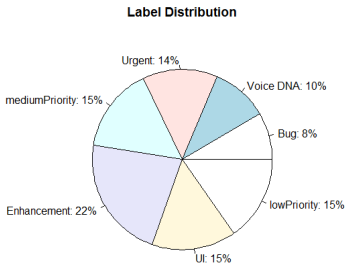

Fig. 9. Label Distribution for Group 2

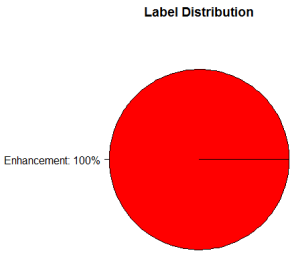

Fig. 8. Label Distribution for Group 1

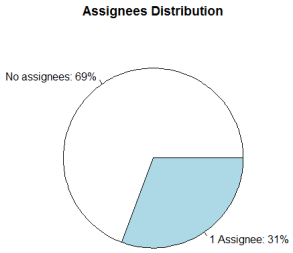

Fig. 10. Assignees Distribution for issues for Group 0

Group No.	No. Closed by Non-Issuer	Total No. Issues	Perc. Closed by Non-Issuer
Group 0	10	32	31.3
Group 1	42	88	47.7
Group 2	8	27	29.6

4) Labels Used: The labels used gives us more information about the issue such as whether the issue is about bug, enhancement or a feature etc. Labelling issues helps in organizing them in a category, thus team members can prioritize them. Hence, it is highly encouraged that the project team labels their issues.

Group 0 used six unique labels with label name such as feature, bug and enhancement etc. which is fine as the number of issues is relatively small at 27. Group 1 used only one label 'enhancement' in the issue. This shows that the Group 1 lacks certian organizational standards which could cause potential problems in their workflow by not having prioritization, considering they have about 32 issues.

On the contrary Group 2 has excellent issue organization system. They have used about 7 labels such as 'mediumPriority', 'urgent' and 'bug' etc. They have about 88 closed issues in their repository and most of the issues are labeled. The use of labels allows them to keep track of issues which needs urgent attention and bugs which needs to be solved.

5) Assignees Distribution: Assignees mentions the list of team members who are assigned for a given issue. By analyzing assignees distribution over all the issues we get to see the work load put onto the team members. We can also assess if the workload is evenly distributed or not. In addition, we can also find out if the issue is assigned at all to any group member.

Group 0 has 69 percent issues which are not assigned to anyone. Rest are only assigned to a single group member suggesting uneven workload distribution. Group 1 has 77 percent of issues which are not assigned, 7 percent assigned only to 1 member and the rest assigned to 2 members of the team. Group 2 has 41 percent of unassigned issues, 35 percent assigned to 1 member, 9 percent assigned to two members and the rest assigned to all three members.

6) User Activity Overall: User activity within an issue may be opening, closing, modifying, adding a label or commenting on an issue. A healthy spread of users interacting with an issue indicates strong team cohesion as each member tracks the project project while keeping all other users informed as well.

Groups 0 and 1 both seem to have team members that do not interact with the issues that often. The activity distribution for Group 2 appears to be quite healthy. Each user interacts with issues at a fairly even amount.

7) Communication within Issues: Issues are extremely useful tool within the scope of a Github repository. Ideally they

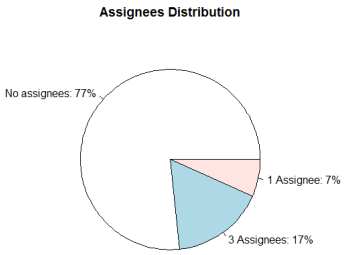

Fig. 11. Assignees Distribution for issues for Group 1

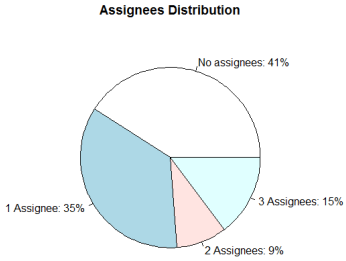

Fig. 12. Assignees Distribution for issues for Group 2

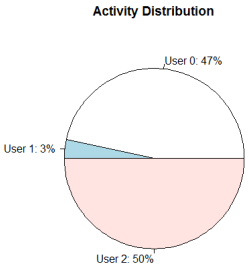

Fig. 13. Issue activity for Group 0

should be used as a record of what is to be completed as well as a marker for team progress. Comments on issues also serve as a visual reminder of what teammates have discussed or how people plan on tackling an issue. In general, the more comments an issue has, the more cohesive the team is.

Group 0 appears to be lacking in regard with the number of comments in each issue. This could be caused by a number of different scenarios including not being familiar with Github. The exact opposite case was for Group1 and Group2 has a lot of interaction can be seen in Figure 16.

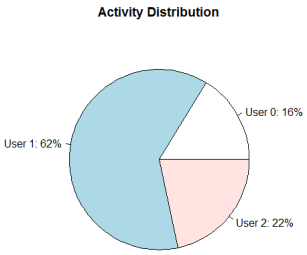

Fig. 14. Issue activity for Group 1

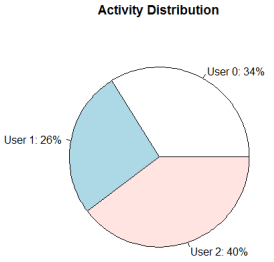

Fig. 15. Issue activity for Group 2

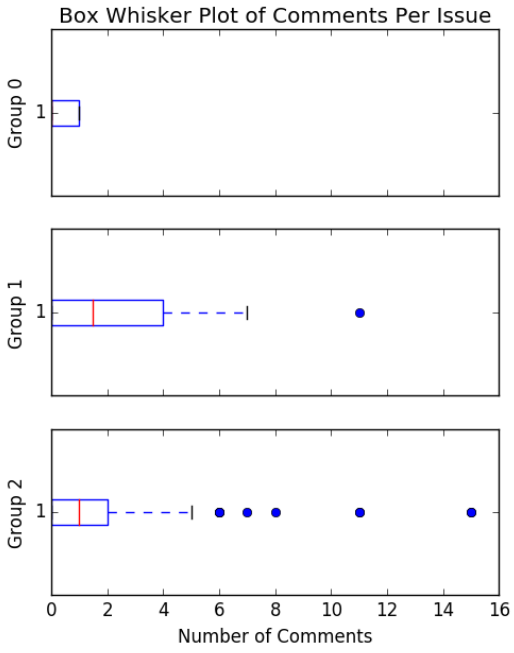

Fig. 16. Inter-issue communication for Group 0

B. Commits

1) Commits per User: Figure 1 details the overall percentage of commits that a given team member has produced. Group

2 shows an unusual pattern as User 0 has contributed just 0.3 percent of all commits. From reading the commits, it seems like he dropped the class early on while only contributing to one commit. We decided to leave this in and not remove it as an outlier since this may happen in real world software engineering projects.

2) Commits per Day: Figure 19 shows a series of box and whisker plots for each team. The number of commits per day are entered in as information. The distribution tends to be pretty low with the majority of number of commits per day being less than five.

The points outside of the box and whisker plot are considered outliers. These are important in our analysis of unusual commit patters later in the paper.

Figure 18 details the number of commits each team has. The X axis shows the number of days since the initial repository creation has passed. The Y axis details information about the number of commits that occurred on the day.

The outliers are especially pronounced within the line graph representation of the commit data as they are shown as towering peaks. We believe that these peaks in data may represent strong team cohesion during development as the development cycle begins to ramp up. Temporal information is important however because late peaks may represent a strained development cycle.

3) Normalized Cumulative Commits Per Day: Figure 20 details the number of commits each team has. The X axis is normalized by each team such that zero is the initial time of the start of the repository while one is that time the final commit was made. The Y axis is also normalized by team such that zero is the initial commit that spawned the creation of the repository while one is the cumulative number of commits.

Since the data is normalized per group based on both the total number of commits and time, this analysis method may be applied to real time, real world software engineering projects as there is no way to determine what the final number of commits will be. This metric is useful for determining the overall pace of development after the fact however.

It is clear to see that Group 0 was falling behind in terms of overall project completion when compared to Groups 1 and 2. Group 1 seems to have done most of their work up front resulting in more than 75 percent of the total commits being made before the first half of the development cycle.

4) Commits Per Day with Milestone Due Dates: Figure 21 is useful to visualize the connection between commit distribution and the due dates of each team's milestones. Overall, there appears to be a pretty loose correlation between the commits that a team makes and the overall due date of milestones.

From personal experience, we feel that teams were quick to abandon milestones since the project was part of a class that already has universal project due dates for all groups. The plot for Figure 22 explores this in further detail.

5) Commits Per Day with Universal Project Due Dates: Figure 22 is formatted in a similar way to that of of Figure 21 except for universal class due dates are shown instead. Class due dates for the January, February and March assignments

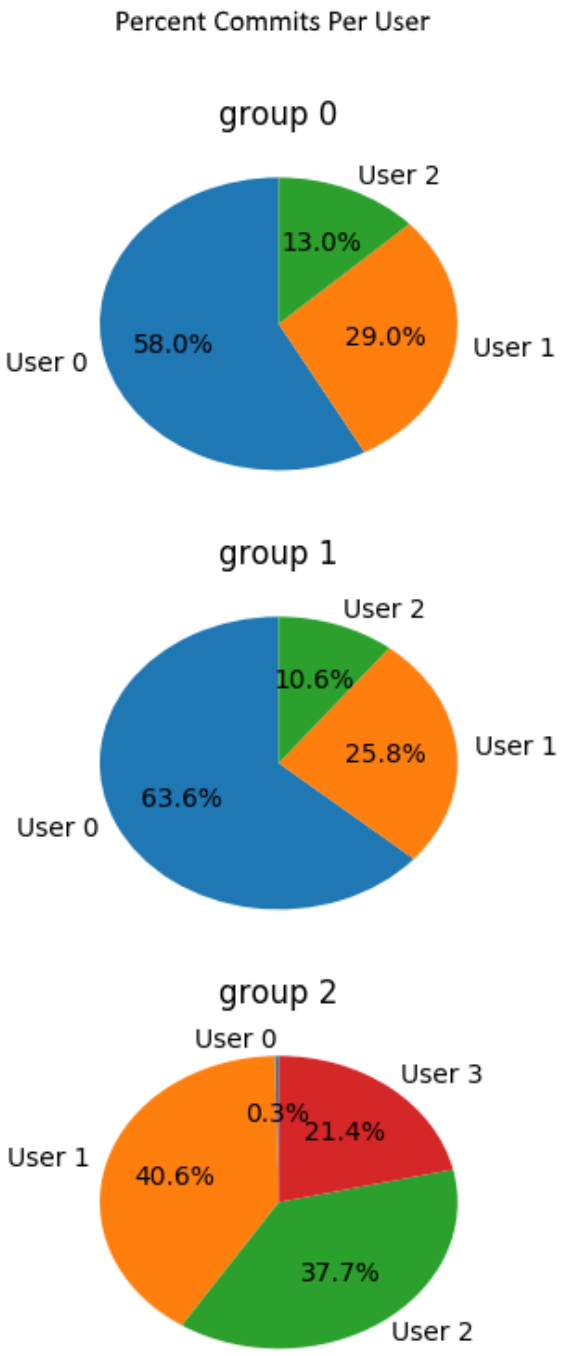

Fig. 17. Pie Charts for each team detailing percent commits per user

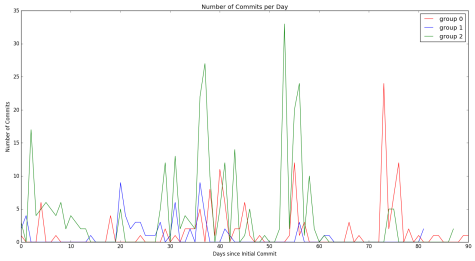

Fig. 18. Plots detailing commits per day for each group

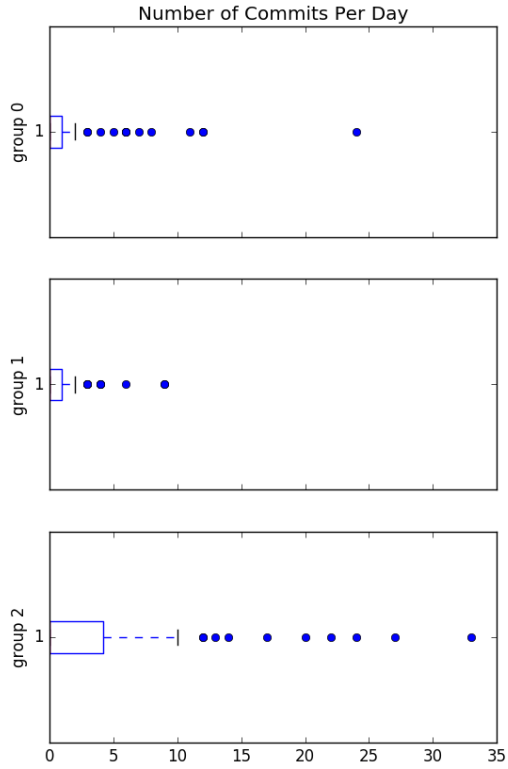

Fig. 19. Box and Whisker plots for each team detailing number of commits per day

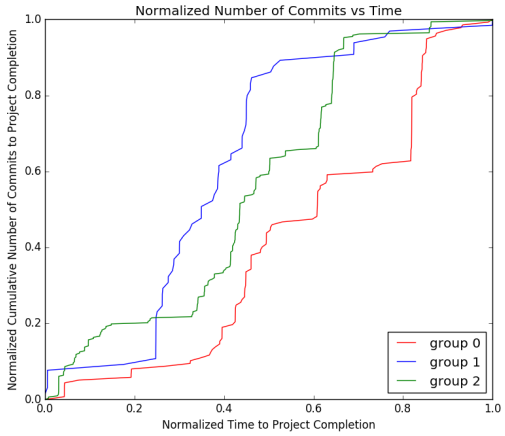

Fig. 20. Plots detailing normalized cumulative number of commits per day for all groups.

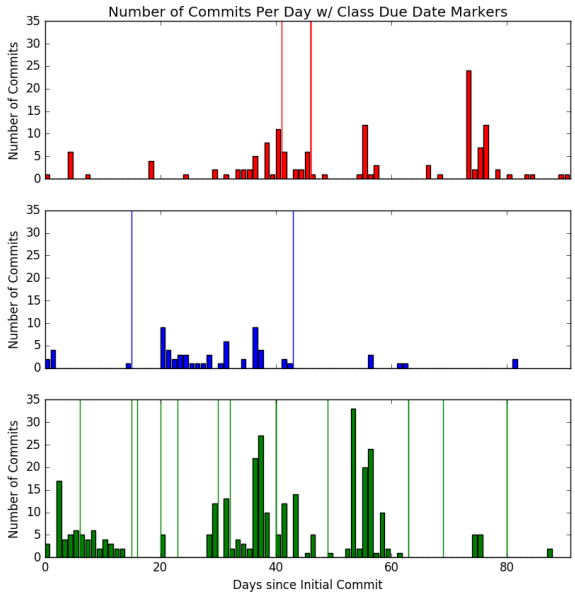

Fig. 21. Plots detailing commits per day with milestone due dates as vertical lines. Red = Group 0, Blue = Group 1, Green = Group 2

are shown as vertical black bars.

IV. Bad Smells

Within this section, an analysis is presented with rearguards to the selection of features that indicate patterns of problematic software engineering practices.

A. Poor Group Communication

One indicator of poor development practices is the lack of group mates participating in issue tracking. Having all group member opening, closing and modifying issues ensures that everyone knows what is to be done in order to advance the development cycle. Group mates that do not take an active part in participating in issue tracking risk not only missing out on important information from other group mates, but also hindering group progress if they do not know the status of what you are working on.

Group 0 showed many signs of this occurring with a lack of balance in issue activity distribution, generally low umber of comments on issues and a low percentage of issues closed by the non-issuer. These factors combined indicate the group had a lapse in communication when it came to issue tracking through Github.

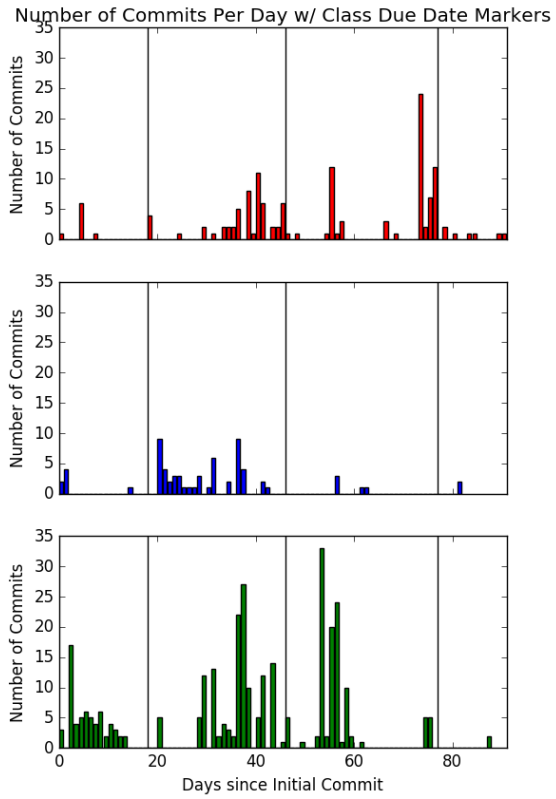

Fig. 22. Plots detailing commits per day with class due dates as vertical lines. Red = Group 0, Blue = Group 1, Green = Group 2

Group No.	High No. Comments per Issue	Diverse Issue Activity	Diverse No. Issues Closed by Non-Issuer	Poor Communication
0	NO	NO	NO	YES
1	YES	NO	NO	YES
2	YES	YES	YES	NO

B. Milestone Misuse

Milestones are an extremely useful tool in the coordination of team activities within Github. Judging by the information collected, teams have been misusing milestones in a few ways.

Group 0 appears to have set milestones midway though their development cycle and forgot about them. Group 0 later closes all of their milestones at the same time once the due date for the overall project is over with.

Group 1 had a few milestones, but ended up never closing them. The only milestone they did close was a milestone that they created on the same day. This is not particularly useful information for a team to have.

Group 2 was one of the only teams that seems to have used milestones correctly. They had a few milestones they were a couple days early for with most milestones being completed a few days after the anticipated due date. Group 2 seems to have put some thought into their use of milestones as a tool for marking progress.

Group No.	Non-Closed Milestones	Milestones 20+ days past due	Milestone Misuse
0	NO	YES	YES
1	YES	NO	YES
2	NO	NO	NO

C. Commit King

In general, the number of commits within a software engineering project should be fairly balanced. Commits are used as a means of tracking group members progress on tasks and keeping the repository up to date at all times.

If one member of a software engineering group contributes to more than 50 percent of the overall number of commits for a project, this may indicate a problem within the group. Judging from the data found in Figure 1, Group 0 and Group 1 both fall victim to this bad smell. Group 2 has a pretty balanced commit workload with no single person making more than 50 percent of the total commits.

Group No.	Greatest User Commit Percentage	Percentage Greater Than 50
0	58	YES
1	63	YES
2	40	NO

D. Late Peaks

Upon analysis of the Box and Whisker plots for commits per day, we realize that teams rarely go over five commits each day. Using this information we consider these days to be indicative of greater patters within the software development cycle. In an of themselves, these high commit days are not a problem, but they may indicate larger issues when coupled with timing information.

From the Normalized Number of Commits vs Time chart, we can see that Group 0 fell behind in terms of progress when compared to Groups 1 and 2. The Number of Commits Per Day with Date Markers shows the large commit per day spikes near the latter half of Group 0's software development cycle. Other Groups did have spikes as well, but they happened pretty early in the development cycle.

Based on this information, we have concluded that large spikes of commits early on in a product development cycle are generally indicate progress, but large commit spikes close to final due days indicate a rocky development.

For ease of analysis in determining peaks, we decided to visualize the number of commits per day as a heatmap. Figure 23 shows the visualization of the number of commits per day with pure red being seven plus commits made in a day. On the opposite end of the spectrum, pure blue represents zero commits made that day.

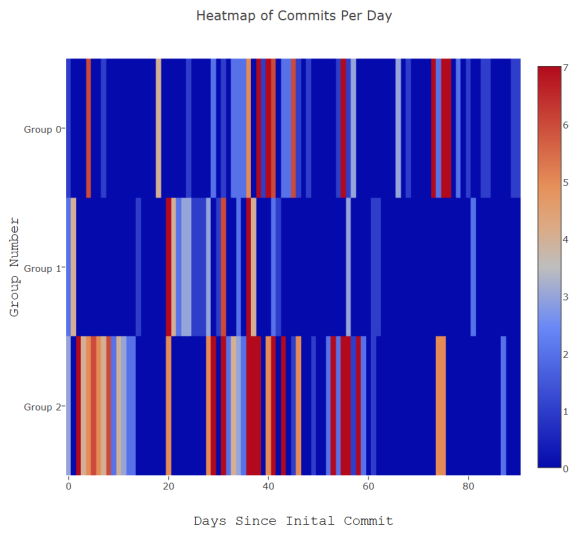

Fig. 23. Plots detailing commits per day as heatmap

For our label, we chose to assign bad smells to groups who have a medium to high number of peaks within the first two thirds of the project (from the beginning of the assignment to the March assignment deadline), and a high number of spikes in the final 1/3rd of the project.

Group No.	Peaks between Jan-Feb due Date	Feb-Mar	Mar-Apr	Late Peaks
0	low	high	high	YES
1	medium	medium	low	NO
2	high	high	low	NO

E. Bad Communicated Work Distribution

The most important part in any software development activity is assigning work to the project members based on their skill and the experience. Failure to do so can result in catastrophic damage to the development of project and great monetary loss.

We observed that all the three groups had issues unassigned with percentage varying from 50 percent to 77 percent. This implies that all the groups failed in planning as to who would solve these issues. Not assigning issues to team members is a trouble maker in the software engineering activity as it would allow team members to sit idle and contribute nothing to the overall project development

Furthermore, for assigned issues there was highly uneven work load distribution. Group 2 had 35 percent issues assigned to only one team member. This number is very high and suggests lack of co-ordination between team members. For all the three groups only small percentage of issues were assigned to two or more team members advocating that the architecture is poorly designed.

Group No.	All Issues Assigned	Diverse Issue Activity	Diverse No. of Commits by Users	Bad Communicated Work Distribution
0	NO	NO	NO	YES
1	NO	NO	NO	YES
2	NO	YES	YES	YES

F. Group Organization

Any software project is developed in small bits of pieces which are then joined together. To accomplish this activity in the most time efficient way possible, work needs to be equally distributed among the team members.

Issues are the metric by which work is split. Having a verity of labeled issues helps with team efficiency. This allows individual team members to focus on the current goals rather than aimlessly wandering around the code wondering what work is to be done. Thus making use of issue labels is a effective way to categorize and prioritizing the work which is essential to any software development activity.

Groupmates should ideally all contribute to issue creation and modification. Having ownership of the creation or modification of an issue ensures that you are actively participating in the organization of the project.

Based on this we say that effective usage of labels on an issue as well as joint ownership of issues significantly improves the overall communication and distribution of workload in software development projects.

Group No.	Diverse Labels	Diverse Issue Activity	Organized
0	YES	NO	NO
1	NO	NO	NO
2	YES	YES	YES

V. Group Comparison using Bad Smell Indicators

After compiling all of the factors that may indicate a Bad Smell, we decided to compared the overall number of bad smells each group triggered. Figure 24 is the graph constructed from this. Group 2 clearly had the least amount of bad smells. The two other groups seemed to have a relatively high number of Bad Smells.

VI. Conclusion

We were successful in gathering Github activity data using Github API and used it to perform analysis on the commits, issues and milestones. We observed few bad smells such as uneven work distribution, ill organization and lack of communication etc. which can be used to identify whether or not the current Github behavior indicates detrimental practices within the development of the project.

In addition, we would also like to make a point that the analysis was done on the class projects and the bad smells may not always reflect the outcome of the project owing to the fact that many students were unaware of the power of using Github as a project management tool.

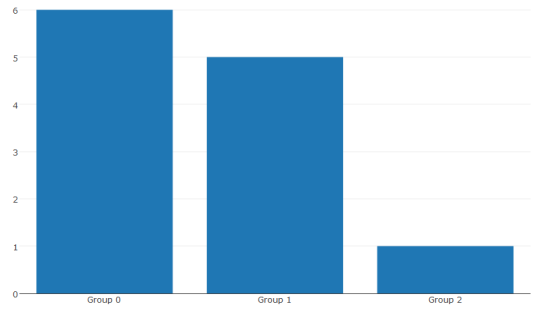

Fig. 24. Group Comparision using Bad Smells Indicators

However, we can say that for software engineering projects where Github or other source control solutions are is actively used, our bad smell indicators would be beneficial to be aware of. Avoiding the outlined Bad Smells would ensure that development goes smoothly as there would be high inter-team communication, good planning and a balanced workload for each developer.

VII. Future Work

It would be interesting to create software which will analyze Github generated data in real time and provide the bad smells for the project along with the suggestion to improve overall workflow. Since Github information is easily obtained using its REST API, software could be made that displays a daily update about the number of Bad Smells your project has.

We also plan to explore few more metrics such as sentiment analysis of the comments which will gives us information about the tone of interaction between team members within comments and commit messages.

There could also be more work done in terms of mining each commit for lines of code changed and file-types added or deleted. This information is not given by the REST API, but web scraping the actual Github repository may be able to obtain those results.